Logic Interface System on CODASYL Database SYSTEM

AUTHOR(S):

Takizawa, Makoto; Itoh, Hideaki; Moriya, Kunihiko

数理解析研究所講究録
593巻 1986年 257-274

Logic Interface System on CODASYL Database SYSTEM

滝沢 誠　　伊藤 秀昭　　盛屋 邦彦
Makoto Takizawa*, Hideaki Itoh**, and Kunihiko Moriya**

*Faculty of Science and Engineering, Tokyo Denki University
**Japan Information Processing Development Center

Abstract

This paper presents the design and implementation of logic language interface on the conventional CODASYL database system. First, we define a formal system of logical aspect of the CODASYL model based on the first-order theory. Second, we make clear the semantics of conventional CODASYL data manipulation language(DML) by defining an abstract machine called a VNM. Then, we show a resolution called a navigational(NV) resolution in which meaningless backtrackings are removed. In the NV resolution, the CODASYL database system is accessed in an interpretive manner, because the CODASYL model provides navigational(record-at-a-time) language DML and resolution is principally based on sequential access.

1. Introduction

Recent computer systems provide various database management systems (DBMS's) which play a central role of various applications. We have now two typical types of commercial DBMS's, i.e. relational[CODD70] and network[CODA73] ones. The relational ones provide non-procedural languages on logical data structure and are used for ad hoc applications like CAD and personal databases. The network ones provide procedural, navigational languages on mixed data structure of logical and physical aspects, and are used for commercial databases because they provide high performance. At present, intelligent non-procedural interfaces on existing CODASYL database systems are needed for supporting ad hoc applications. In addition to the conventional DML interface, the CODASYL database systems can provide non-procedural relational interfaces[TAKI80, DAYA82]. Logic languages like Prolog[KOWA79] provide more descriptive power, i.e. recursive and non-deterministic view definitions, but relational languages like SQL[DATE81] can provide only non-recursive and deterministic definitions of views. Logic interfaces on the

relational databases have been discussed by [CHAN81, LID84, OHSU83, etc.].

In this paper, we present logic language interface on existing CODASYL database systems. First, we separate the conventional CODASYL model into logical and physical models. For the logical one, we define a formal system based on the first-order theory. For the physical one, we make clear the semantics of conventional DML by defining an abstract machine called a VNM. Then, we propose a new refutation procedure called a navigational refutation(NVR) procedure which avoids meaningless backtracking and in which the CODASYL database is accessed interpretively by using the DML's.

In chapter 2, we define a formal system called a conceptual network system(CNS) and define the semantics of DML. In chapter 3, we present our NV refutation procedure.

2. CODASYL Model

In the conventional CODASYL model[CODA73, OLLE78], both physical and logical aspects are mixed. In order to put the logic interface on the CODASYL database system, both aspects have to be explicitly separated.

2.1. Conceptual Network Data Structure

By abstracting the logical aspect from the CODASYL data structure, we define a conceptual network data structure **N** composed of sets and partial functions on these sets. **N** is a pair of a schema S which is a time-invariant, logical data structure, and a database which is a set of time-variant data. The schema is composed of record(R)-types and set(S)-types. An R-type A is a set of items {@A, t1,..., tm} ($m \geq \emptyset$), where @A is a database(db)-key and ti a data item($i=1,\ldots,m$). For every item t in A, let dom(t) denote its domain. A record occurrence(RO)-set A for A is A $\subseteq$ dom(@A)x dom(t1)x...x dom(tm). Each a $\in$ A is a record(R)-occurrence. For each item t, let t(a) denote t's value of a. Every a in A has a different value @A(a).

When a partial function from an RO-set A to B exists, a set(S)-type C =(A, B) is defined. Here, A is an owner and B a member. Also, A and B are distinct. A set occurrence(SO)-set C for C is defined as C $\subseteq$A x B, which is a partial function C:A->B. Each element <a,b> in C is an S occurrence.

A collection of RO-sets and SO-sets is a conceptual network database(CDB). Fig. 1 shows an example of a graph representation of a conceptual network schema on information of

departments D, employees E, and projects P, where boxes D, E, P, EPL represent R-types and arcs DE, PE, and EP S-types.

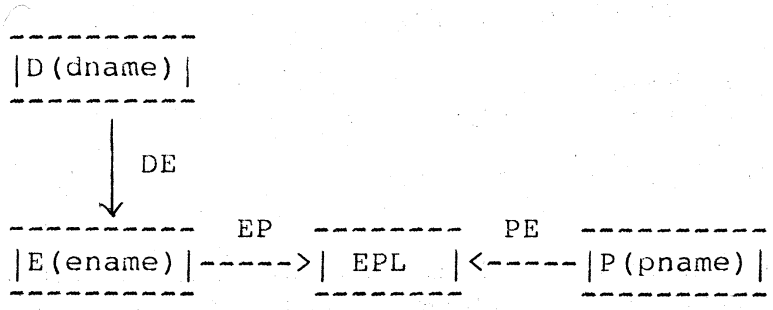

Fig. 1 Conceptual network schema.

2.2. Conceptual Network System

Based on the conceptual network data structure, we define a formal system named a conceptual network system(CNS) as a quadruple <S, L, G, N> where S is a conceptual network schema, L a language, G axioms, and N inference rules. L for S is a first-order language[ENDE72] named a conceptual network language(CNL), whose symbol set A includes four kinds of predicate symbols, i.e. T, S, R, and V ones. For every item, R-type, and S-type in S, L includes a T-, R-, and S-predicate symbol, respectively. Other predicate symbols are V-types. Terms and well-formed formulas(wffs) are defined in the same way as [ENDE72].

Now, an interpretation I of L is a pair <D, M>. D is a universe. M is a mapping from the symbol set A where, for each constant a, M(a) $\in$ D, for each n-ary function symbol f, $M(f):D^n \rightarrow D$, and for every n-ary predicate symbol P, $M(P) \subseteq D^n$. Here, let U, A, C be T, R, S symbols, respectively. M(U) denotes a domain of item, M(A) an RO-set, and M(C) an SO-set. A collection of M(E) for every T or R symbol E is a conceptual network database(CDB). Satisfiability of wff W, i.e. =W, is the same as [ENDE72].

The set G of axioms contains the following nonlogical axioms.

(1) For every (m+1)-ary R-symbol A for R-type A =(@A,I1,...,Im),

$\forall k \forall x1 \ldots \forall xm$
(A(k,x1,...,xm)-> @A(k) and I1(x1) and ... and Im(xm)).

$\forall k \forall x1 \ldots \forall xm \forall y1 \ldots \forall ym$ (A(k,x1,...,xm) and
A(k,y1,...,yn) -> (x1=y1) and...and (xm=ym)).

(2) For every S-symbol C for S-type C =(A, B),

∀k ∀h (C(k, h) -> @A(k) and @B(h)).

∀k ∀h xl... xn yl... ym
(C(k, h) -> A(k,xl,...,xn) and B(h,yl,...,ym)).

∀k ∀h ∀j (C(k,h) and C(j,h) -> k=j).

[KOBA85] also gives more general axioms for various data models. Interpretations which satisfy these nonlogical axioms are models of the CNS. Modus ponens and generalization are inference rules of the CNS. For a wff W, -W represents that W is a theorem, i.e. W is derivable by applying the inference rules on the axioms. Since the CNS is a first-order theory, it is obvious the following proposition holds.

[Prop. 1] For a wff W in the CNS, - w if and only if = w. ■

2.3. Virtual Network Machine (VNM)

We define a virtual network data structure **V** by introducing ordering relations on the conceptual network data structure **N**. We also define a virtual network machine (VNM) which executes virtual operations (VOP's) on **V** which correspond to the DML's[OLLE78].

2.3.1. Virtual Network Data Structure

A virtual network database(VDB) is a collection of virtual RO(VRO)-sets and virtual SO(VSO) sets. For every RO-set A in the CDB, a VRO-set X is defined as a totally ordered set (A, <) where, for every a and a' ∈ A, a < a' if and only if @A(a) < @A(a'). Each element in X is a VR occurrence.

For every SO-set C of S-type C=(A,B), a VSO-set Z is defined as a set {<a,m(a)>| a∈A and m(a)≠ϕ}. Here, m(a) is a set of all a's members in B. It is totally ordered in either an ascending/descending order on some item of B or arbitrary order. Each element in Z is a VS occurrence. VRO-sets for A and B are an owner and member VRO-sets, respectively.

2.3.2. Virtual Network Operations (VOP's)

A virtual network machine(VNM) is composed of registers a1, ..., an, g, dØ, and for every VRO set X and VSO set Z, dX and dZ, respectively, where dØ, dX, dZ called currencies correspond to currency indicators[OLLE78]. The VNM manipulates the VDB by the following virtual operations(VOP's). Here, let X and Y be

member and owner VRO sets for Z, respectively.

(1) first (X)/ last (X). The db-key of the first/last VR occurrence in X is stored in dØ and dX. Here, a register d is said to denote VR occurrence x if and only if x's db-key is stored in d. Let (d) be a VR occurrence x denoted by d.

(2) next (X)/ prior (X). The db-key of the VR occurrence next/prior to (dX) is stored in dØ and dX.

(3) first (Z)/ last (Z). The db-key of the first/last VR occurrence in m(x) where x = (dX) is stored in dØ, dY, and dZ.

(4) next (Z)/ prior (Z). The VR occurrence's db-key next/prior to y in m(x) where y = (dZ) is stored in dØ, dY, and dZ.

(5) owner (Z). The db-key of x in owner X where (dZ) ∈ m(x) is stored in dØ and dX.

(6) get (X, a). The data item values of x ∈ X where x = (dX) is stored in register a.

(7) any (X, v). If some item t of X is a direct access item, the db-key of x ∈ X is stored in dØ and dX, where t(x) = v and there exist no x"∈X such that x"<x and t(x")=v.

(8) dup (X, v). If t is a direct access item, the db-key of x' in X is stored in dØ and dX, where x<x', t(x')=v, and there exist no x" in X such that x<x"<x'.

(9) find (X, a). The db-key of x ∈X which is stored in register a is stored in dØ and dX.

(1Ø) accept (X, a)/ accept (Z, a). The db-key in dX/dZ is stored in register a.

If VOP's execution terminates correctly, "S(uccess)" is stored in the register g, otherwise "F(ailure)". It is easily understood that for each VOP there exists a DML. For example, find next Y within Z for next(Z), and find X; db key is a for next (Z).

3. Navigational Resolution

Let us discus how to resolve CNL clauses by using the VOP's.

3.1. Clause Form in the CNL

We assume that every wff in the CNL is in a Horn clause [KOWA79]. There are two kinds of Horn clauses, i.e. definite and goal ones. A definite clause W is in a form A<-B1, ..., Bm(m≥0), where A and Bi(i=1,...,m) are atoms of a form R(t1,...,tn) where R is an n-ary predicate symbol and ti a term (i=1,...,n). A is a head, and B1,...,Bm a body. If m = 0, W is a unit clause, otherwise, a rule clause. A goal clause G is in a form <- B1,...,Bm(m≥0). If m = 0, G is an empty clause□. Variables prefixed by ? in G are target variables. Variables are written in upper-case letters and constants lower-case ones.

Let us give you an example of a definite clause set F for the schema in Fig.1.

```
F = { 1) ED(N,M)<- E(X,N),DE(Y,X),D(Y,M).
      2) EE(N,M)<- E(X,N),EP(X,Y),PEL(Y),PE(Z,Y),PE(Z,M).
      3) PP(N,T)<- EE(N,T).
      4) PP(N,T)<- EE(N,U),EE(M,U),PP(M,T).
      5) E(1,a)<-.      6) E(2,b)<-.
      7) E(3,c)<-.      8) E(4,d)<-.
      9) D(1,c)<-.     10) D(2,a)<-.
     11) DE(1,1)<-.    12) DE(1,3)<-.
     13) DE(2,2)<-.    14) DE(2,4)<-.
     15) P (1,d)<-.    16) P (2,n)<-.
     17) PE(1,1)<-.    18) PE(1,2)<-.
     19) PE(1,3)<-.    20) PE(2,4)<-.
     21) PE(2,5)<-.    22) EP(1,1)<-.
     23) EP(1,4)<-.    24) EP(2,5)<-.
     25) EP(3,2)<-.    26) EP(4,3)<-.
     27) EPL(1)<-.     28) EPL(2)<-.
     29) EPL(3)<-.     30) EPL(4)<-.
     31) EPL(5)<-.   }          ...(3.1)
```

Here, E, D, P, and PEL are R-symbols, DE, PE, and EP are S-symbols, and DE, EE, and PP are V-symbols. Unit clauses 5)~8) denote that a,b,c,d are employee, 9)~10) that there are two departments c and a, 11)~14) that department c has members a and c, and a has b and d, 15)~16) show there are two projects d and n, and 17)~31) indicate that project d has members a, c, d, and n has a, b, e. Rule 1) indicates that employee N belongs to department M. Rule 2) indicates that employee N belongs to project M. Rules 3) and 4) show that N is an employee who participates in project T including T's members M and employees who belong to the same projects as M.

3.2. SLD Resolution

Let P be a set of definite clauses and G a goal clause <-B1,...,Bm. Suppose that a computation rule R[LLOY85] gives an atom Bi called a selected atom from G. A substitution θ is in a form {X1/t1,...,Xh/th} where Xi is a variable and ti a term(i=1,...,h). For an expression E, Eθ is one obtained by replacing simultaneously all occurrences of Xi in E by ti(i=1,...,h). Let C be an input clause B<- A1,...,Ak in P, and θ a substitution such that Bθ=Biθ. An SLD resolvent of G and C with θ is a goal <- (B1,...,Bi-1, A1,...,An, Bi+1,...,Bm)θ. An SLD deduction of P U{G} via R is a sequence of goal clauses Gθ(=G),G1,...,Gn, input clauses C1,...,Cn, and substitutions θ 1,..., θn, where Gi is an SLD resolvent of Gi-1 and Ci with θi (i>1). An SLD refutation F is an SLD deduction which derives an empty clause. An answer substitution is a restriction of composition θ1...θn in F on the target variables in G.

Let us take a goal G= <-PP(?N,d) and F of (3.1) as an example. All possible SLD deductions of F U {G} via some computation rule R are represented by an SLD tree as shown in Fig.2. Here, nodes denote goals and selected atoms are underlined. Labels attached on branches show input clauses in F. For example, a goalG1= <-EE(?N,d) is derived from G and the input clause 3). Each root-to-leaf path denotes an SLD deduction. Paths denoting refutations are success paths.

Next problem is how to find success paths in the SLD tree. Most Prolog systems[CLOC84] use a depth-first search of the SLD tree using a prefixed ordering of input clauses. Let us consider a goal G2= <-E(X,?N),EP(X,Y), PEL(Y), PE(Z,Y),P(Z,d) in Fig.2, where an atom PEL(Y) is selected by R. First, an input clause 27) is selected and a resolvent G3= <-E(X,?N),EP(X,Y), PE(Z,Y), P(Z,d) is derived. Then, a resolvent G4= <-E(X,?N),EP(X,Y), P(Z,d) is derived. Like this, an empty clause is derived.

Now, let us consider another example of a goal G = <-A(X,Y),B(Y),C(X). By the Prolog refutation procedure, first, A(X,Y) is selected and a resolvent G1= <-(B(Y),C(X))θ1 is derived, where θ1 is a substitution for variables X and Y. Then, a resolvent G2= <-C(X)θ1θ2 is derived from G1, where θ2 is a substitution for Y. But, suppose that the resolution of G2 fails. We go back to G2 and try to find a next input clause. This process is called a backtracking from G2 to G1. Here, looking at G1, atoms B(Y) and C(X) have no common variables. This implies that the backtracking from G2 to G1 is meaningless since further resolutions of <-B(Y) do not generate any new substitutions for variables in C(X). Thus, the Prolog procedure cannot prevent those meaningless backtrackings.

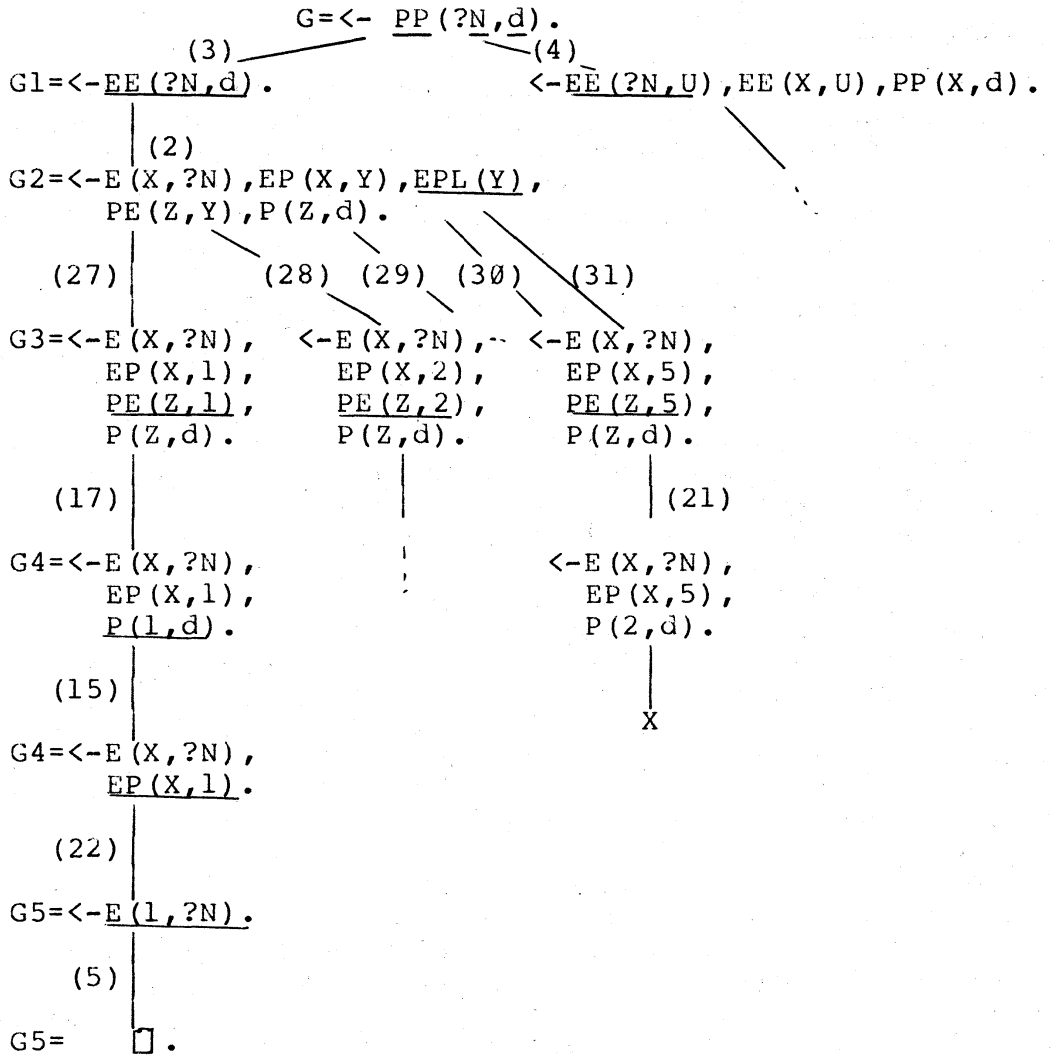

Fig.2 SLD tree.

3.3. Navigational Refutation (NVR) Procedure

Now, let us present our navigational refutation(NVR) procedure which aims at improving the Prolog procedure so as to take advantage of the virtual network machine (VNM), i.e. record-at-a-time manipulation of data, and increase the performance. First point on considering a logic interface on the VNM is that a definite clause set P includes very more unit clauses whicn denote R- or S-occurrences in the conceptual network database (CDB) than rules. Next, it is required to get all answer substitutions for a goal. Furthermore, it takes for resolution much CPU time that is comparable with I/O time. A backtracking to some goal G requires that G be resolved for next input clause C defined by some prefixed ordering. This backtracking consumes computer resources. If the number of backtackings is decreased,

the refutation performance can be improved. So, intelligent backtracking[CAMP84] has been proposed. Here, we propose a partially-compiled method that once an SLD resolution F is found, a VOP program which derives whole result from the VNM is synthesized with respect to the sequence of the selected atoms in F. Also, we prevent meaningless backtrackings by going back from a goal G to G'

Now, let S be a set of success paths, i.e. SLD refutations, in an SLD tree T for P U {G} via a computation rule R. Problem is how to find all the success paths in T. In the conventional Prolog systems, once a success path is found, another one is tried to be found by triggering backtrackings. As we stated before, resolutions consume computation resources, i.e. CPU and main memories. Our approach toward decreasing the number of the resolutions to get all success paths in T is that once a success path, i.e. an SLD refutation F, is found, we generate a VOP program for F, which can derive all answer substitutions for all refutations F' similar to F. By this, the resolutions for F' can be avoided, What is the similarity relation between the refutations?

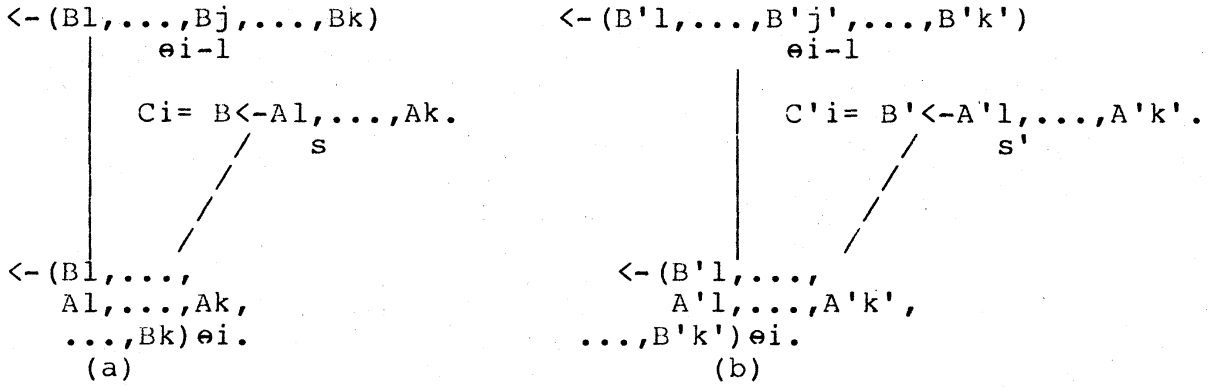

Fig.3 Equivalence resolution.

Now, let us define a similarity relation ≅ on S.

[Def.] Let F and F' be SLD refutations for P U {G} via R. F and F' are said to be similar (F≅F') if and only if the following conditions hold.

1) F and F' have the same length n.

2) For every i(=0,1,...,n)-th resolution in F and F' as shown in Fig.3 (a) and (b), respectively, the followings hold.

2-1) θi= θi-1 s if i>0, e if i=0. θ'i= θ'i-1 s' if i>0, e if i=0.

2-2) k=k', j=j'(Bj=Bj') m=m'. If m=m'=∅, B and B' are atoms of the same predicate symbol. If m=m'>∅, the input clauses Ci ad Ci' are the same .[]

It is clear that the similarity relation ≅ is an equivalence relation. Hence, S is divided into equivalence classes, S1, S2,..., with respect to ≅, where Si = {F| F≅F' and F, F'eS}. In two refutations F and F' in S, when the input clauses are ground unit clauses, only their occurrences are different. For example, in the SLD tree in Fig. 2, the success paths are divided into equivalent classes S1= { (3)-(2)-(27)-(17)-(15)-(22)-(5), (3)-(2)-..., ..., (3)-(2)-(31)-(21)}, S2= {(4)-...,....}, S3={(4)-...},....

Next, once an SLD refutation F is found, how to find all similarity refutations. Now, let B be a sequence of selected atoms in F. Let A be a subsequence obtained by omitting from B ones whose input clauses are rule clauses. That is, A contains only R-, S-, and N-atoms. A following SLD refutation F" of P U {A} is defined, i.e. selected atoms are selected in an order of A. In F", input clauses are ground unit clauses, i.e. R- or S-occurrences. For example, in S1, A is a goal <- EPL(Y),PE(Y,Z),P(Z,d),EP(X,Y),E(X,?N). Input clauses are selected with respect to the ordering defined in the VNM. From this refutation, we generate a VOP program as shown in Fig.4.

```
      first (EPL).   go to M1.
L1:   next  (EPL).
M1:   if ( g = F)    go to END.

      owner (PE).    go to M2.
L2:                  go to L1. /* backtrack */
M2:   if ( g = F)    go to END.

      get(P).
      if ( P.pname~=d) go to L2.
                     go to M3.
L3:                  go to L2. /* backtrack */
M3:

      owner  (EP).
L4:                  go to L3.
M4.   if ( g = F)    go to L3. /* backtrack */

      get(E).
      output(E.ename). go to L4. /* backtrack */
```

Fig.4 VOP program.

It is clear that all answer substitutions in the equivalence class S2 are derived by this VOP program. But, this program

includes meaningless backtracking.

3.4. Optimization

Then, we try to generate a VOP program V for an equivalent class Si, which does not include meaningless backtrackings.

First, we define a goal graph. For an atom A, let V(A) be a set of variables in A. For atoms C and D, we define an operation named a substitution intersection C^D on θ as a set $\{\langle x,y\rangle \mid x\theta\in V(C),\ y\theta\in V(D),\ \text{and } x\theta=y\theta\}$.

[Def.] For a goal <-B1,...,Bm and a substitution θ, GL({B1,...,Bm}) is a goal graph obtained by the following procedure:

1) For every subgoal Bi, a node Bi is created.

2) For every pair of nodes Bi and Bj, for each <x,y> in V(Bi)^V(Bj) on θ, an edge <Bi:x, Bj:y> between Bi and Bj is created. □

[Def.] Let P be a set of definite clauses, G a goal, R a computation rule, and F an SLD refutation of P U{G} via R. Let M be a set $\{\langle \bar{A},\tilde{\theta}\rangle \mid \bar{A}$ is an R or S atom in P U{G} and A is a selected atom with a substitution $\tilde{\theta}$ where $A=\bar{A}\tilde{\theta}$ in F$\}$. A target graph for F is a goal graph of atoms in M. □

Now, we define a navigational(NV) tree.

[Def.] A navigational(NV) tree T for a goal graph G is defined as a tree which satisfies the following conditions:

1) There exists a bijection from nodes in T to nodes in G.

2) If there is an edge between nodes X and Y in G, X and Y are in the same path in T.

3) Children of every node X are totally ordered in a right-to-left order, i.e. nodes in T are totally ordered in a depth-first manner.

A target node <B,θ> is a first node among ones in T which include any variable X for some target variable Y such that $X\theta=Y\theta$. A target subtree in T is a subtree which includes any target node. □

Fig.5 shows an example of an NV tree for a goal G1 in Fig.2. A node (1) is a root and (2) a target node.

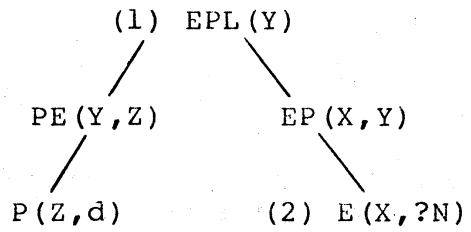

Fig.5 Navigational tree.

Now, we present a navigational refutation(NVR) procedure to find an SLD refutation.

[NVR] A navigational(NV) refutation of P U {G} via R consists of sequences of goals G∅(=G), G1,...,Gn(=[]), NV trees T∅,...,Tn, goal graphs GL∅,...,GLn, input clauses C1,...,Cn, and substitutions θ1,..., θn if the following conditions hold:

1) T∅=ϕ, θ∅=e(identity substitution), GL∅=GL(G∅).

2) Let Gi-1 be <-B1,...,Bm. Suppose that R gives a selected atom Bj and a node X in Ti-1 from Ti-1, Gi-1, and θi-1, i.e. <Bj, X>=R(Ti-1, Gi-1, θi-1). Let Ci be B<-A1,...,Ak (k≥∅) with a substitution σ̃ where Bσ̃=Biσ̃. Gi is an SLD resolvent of Gi-1 and Ci with σ̃. θi=θi-1. There are two cases on k.

a) In the case of k=∅, a VRO or VSO set denoted by B is accessed navigationally by VOP's to find an occurrence which satisfies θi-1. Ti is obtained by adding a node <Bi, θi-1> as a last child of X in Ti-1. GLi is obtained by deleting from GLi-1 a node Bi and edges whose both ends do not exist.

b) k≥1. GLi is obtained by deleting Bi and all edges incident to Bi from GLi-1, and adding GL({A1,...,Ak}) to GLi-1. Ti=Ti-1.

Here, θn is a navigational(NV) substitution. □

[NV rule] For a tree T, a goal graph GL, and a substitution θ, a navigational(NV) rule R gives a selected atom B in GL and a node X in T by the following procedure where cost functions fcost and ecost are defined in a later section:

(1) Let X be a right-most leaf in T.

(2) If GL=ϕ, go to (6).

(3) Select a node B in T such that ecost(B,θ) is minimum and V(B)^V(X)≠ϕ on θ.

(4) If found, return B and X.

(5) Otherwise, if X is not a root, then let X be a parent of X and go to (2).

(6) Select a node B in GL whose fcost(B,θ) is minimum and return B and X. □

[Prop. 2] A sequence of Gθ,...,Gn, C1,...,Cn, θ1,..., θn in the NV refutation is an SLD refutation.

[proof] By the definition, it is obvious. ■

This proposition implies that our system based on the NVR procedure is sound and complete because of soundness and completeness of the SLD resolution[LLOY85].

[Prop. 3] An NVR procedure generates an NV tree.

[proof] Let G be a target graph of an SLD refutation F. By the definition, a tree Tn in the NVR procedure is generated by searching G in a depth-first manner. In Tn, all edges in G are contained in the same path. ■

[Prop. 4] Let T be an NV tree, θ an NV substitution for T, S a subtree of T, and θ' an NV substitution for S where there is θ" such that θ= θ'θ". If there exists another NV substitution θ̃ of S for given θ", θ̃θ" is also an NV substitution for T.

[proof] By the definition, every two nodes which are included in different paths in T do not have common unifiable variables. So, θ" does not contain substitution for variables of nodes other than in S. ■

This proposition means that once an answer substitution is found, subtrees which contain no target nodes can be skipped when backtracking.

3.5. VOP Program for a Navigational(NV) Tree

A VOP program V for an NV tree T is a set of cells interconnected by directed links. For every node X in T, there exists a cell C(X) which has two input ports, F(X) and N(X), and four output ports, S(X), B(X), PT(X), and NT(X)[see Fig.6]. C(X) is considered as an object which contains an ordered set D(X), i.e. VRO-set or VS-occurrence, and manipulates D(X) by the VOP's. Links represent output-to-input relations among cells. There are four kinds of links, S, B, PT, and NT. Let X and Y be nodes in T. If Y is next to X in a depth-first order of T, an S-link exists from S(X) to F(Y). If Y is an X's parent in T, a B-link exists from B(X) to N(Y). If Y is a target node which is

left-/right-nearest to X in T, a PT-/NT-link exists from PT(X)/NT(X) to F(Y).

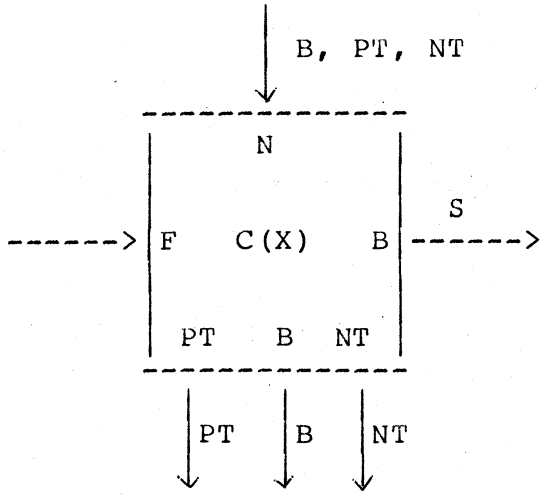

Fig.6 Cell structure.

C(X) is activated on receipt of message token at an input port, F(X) or N(X). Substitution θ is sent via the S-link. On receipt of θ at F(X), D(X) is sequentially accessed from the first occurrence in D(X) by the VOP's. If an occurrence x which satisfies θ is found, the currency of x is saved, substitution θ' is constructed from x, and send a token with θθ' via S(X). If not found, a token is sent via B(X). On receipt of a token at N(X), D(X) is sequentially accessed from the occurrence denoted by the saved currency in C(X).

Fig.7 shows an example of a VOP program of an NV tree T of Fig.6. Using the proposition 4, once the first answer substitution is found and T is constructed, only target subtrees of T are accessed. By this, generation of substitutions independent of the answer substitution is avoided. When all answer substitutions from T are obtained, i.e. a token is output from B(X) of the V's first cell X, a backtracking occurs to a first goal among ones in the NV refutation whose input clause is a rule with no target variables. Also, Fig.8 shows a cell structure of the VOP program of Fig.4.

```
      first (EPL).        go to M2.
L1:   next  (EPL).
M1:   if(g=F)             go to END.
      owner (PE).
      if(g=F)             go to L1.
      get   (P).
      if(P.pname~=d)      go to L1.
```

```
owner (EP).
if(g=F)            go to L1.
get   (E).
output(E.ename).   go to L1.
```

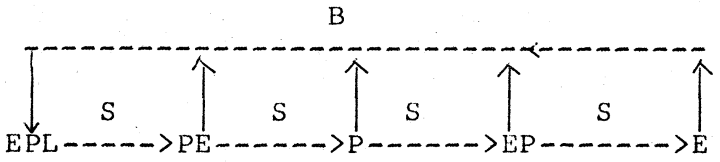

Fig.7 Vop Program.

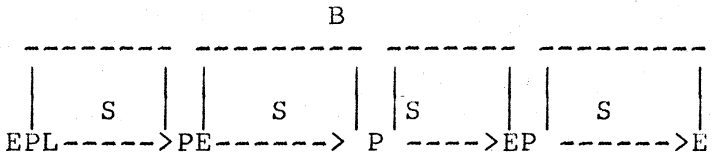

Fig.8 Vop Program of Fig.4.

3.6. Cost Functions

Let us define the cost functions fcost and ecost used in the NV rule R. Assume that cost measure is an expected number of VR occurrences manipulated by the VNM. fcost(B,θ) gives a cost for getting all substitutions for an atom Bθ. There are two cases. First case is B denotes a VRO set X. Let |X| be an X's cardinality and st a selectivity[HEVN78] of an item t in X.

(a) fcost(B,θ)=st |X| if, for a variable T to the item t, Tθ=c and t is a direct access item.

(b) fcost(B,θ)=|X| otherwise X is sequentially accessed.

Next, B is a V atom and V<-C1,...,Cn ia an input clause.

(c) fcost(B,θ) = M Summation of cost(Ci) for i=1,..,n.

If B is not recursive, M=1, else M $\geq$ 1. By increasing M, the resolution of recursive views can be delayed.

ecost(B,θ) gives a cost for getting all answer substitutions of Bθ. Let Z be a VSO set denoted by B if B is an S atom.

$$ecost(B,\theta)=\begin{cases}\text{.the average number of member VR occurrences which each owner has}(\geq 0) \text{ if sequentially accessed by first(Z) and next(Z).}\\ \text{.the possibility that each member has an owner } (\leq 1) \text{ if accessed by owner(Z).}\\ \text{.fcost}(B,\theta) \quad \text{otherwise.}\end{cases}$$

4. Concluding Remarks

Fig.9 shows an overview of our system called LIP(Logic Interface Processor on the CODASYL database system). The LIP is implemented in Lisp. Each VOP is called via a LISP function call, each of which is written in PL/I. At present, Fujitsu's AIM/DB of a CODASYL type is used as a fact database which stores ground unit clauses. The rules and new facts are stored in a relational database system AIM/RDB. The AIM/DB is accessed by using Lisp VOP functions in an interpretive manner. Our LIP is operational on Fujitsu's M-360R.

In this paper, we showed the first-order theory of logical part of the conventional CODASYL model, and made clear the semantics of the CODASYL DML by the behavior of the abstract machine VNM. Next, based on the formal system, we showed how to translate logic queries into DML operations. Our system accesses the CODASYL database by interpretive manner because the VOP, i.e. DML, navigationally manipulates the data in the database. Although a lot of researchers have tried to implement logic database using the relational model, major parts of existing databases are of a CODASYL type, and we think that the CODASYL model is also in harmony with logic.

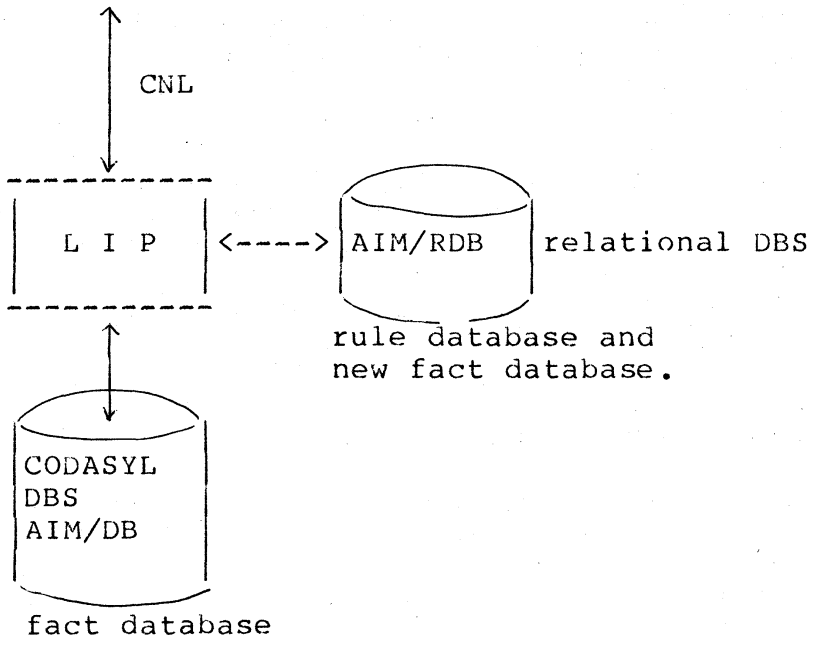

Fig.9 Overview of LIP.

References

[APT82] Apt, R., and van Emden, M.H., "Contributions to the Theory of Logic Programming," JACM, Vol.29, No.3, pp.841-862.

[CAMP84] Campbell, J.A., "Implementations of Prolog,"Ellis Horwood Limited, pp.175-278, 1984.

[CHAN81] Chang, C.L., "On Evaluation of Queries Containing Derived Relations in a Relational Data Base," Logic & Database, Plenum Press, 1981.

[CLOC84] Clocksin, W. F. and Mellish, C. S., "Programming in Prolog(2nd ed)", Springer-Verlag, 1984.

[CODA73] CODASYL DDL Committee, "CODASYL Data Description Language," Journal of Development, 1973.

[CODD70] Codd,E. F., "A Relational Model of Data for Large Shared Data Bank," CACM, Vol.13, No.6, 1970, pp.337-387.

[DATE81] Date, C.J., "An Introduction to Database Systems," Addison-wesley, 1981.

[DAYA82] Dayal, U., et al., "Query Optimization for CODASYL Database Systems," Proc. of the ACM SIGMOD, 1982, pp.138-150.

[ENDE72] Enderton, H, "Mathematical Introduction to Logic," Academic Press, 1972.

[HENSL84] Henschen, L.J. and Naqvi, S.A.,"On Compiling Queries in Recursive First-Order Databases," JACM, Vol.31, 1984, pp.47-85.

[HEVN78] Hevner, A. and Yao, S.B., "Query Processing on a Distributed Databases," Proc. of the Third Berkeley Workshop, 1978, pp.91-107.

[JACO82] Jacob, B.E., "On Database Logic," JACM, Vol.29, No.2, 1982, pp.310-332.

[KOBA85] Kobayashi, I., "Classification and Transformations of Binary Relationship Schemata," Sunno Institute of Business Administration, 1985.

[KOWA79] Kowalski, R., "Logic for Problem Solving," North-Holland, 1979.

[LID84] Li, D., "A Prolog Database System," Research Studies Press, 1984.

[LLOY85] Lloyd, D., "Foundation of Logic Programming," Springer-Verlag, 1984.

[OHSU83] Ohsuga, S., "Developing a Deductive Relational Database for Uniform Handling of Complex Queries," JIP of IPSJ, 1983, pp.123-137.

[OLLE78] Olle, T., "The CODASYL Approach to Data Base Management," John Wiley & Sons, 1978.

[SMIT85] Smith, D.E. and Genesereth, M.R., "Ordering Conjunctive Queries," Artificial Intelligence, Vol.26, 1985, pp.171-215.

[TAKI80] Takizawa, M. et al.,"Query Translation in Distributed Databases," IFIP'80, Tokyo-Melbourn, 1980, pp.451-456.

[ULLMJ85] Ullman, J.D., "Implementation of Logical Query Languages for Databases," TODS, Vol.10, No.3, 1985, pp.289-321.